

Software Quality Assurance Project Report

Authors - Jack Woods, Dylan Stancil, Devin Spivey

Team Name – BackLeft

GitHub Link - [jtw0086/BackLeft-SQA2022-AUBURN \(github.com\)](https://github.com/jtw0086/BackLeft-SQA2022-AUBURN)

Activities Done

1. Jack set up the initial GitHub repository and created a README.md with the required information, completing tasks 1 – 3.
2. The team delegated each individual task under Activity 4 to each member of the team.
 - a. Dylan chose to work on the Git Hook task (4a) and created a new branch (branch 1) to do this work on.
 - b. Devin chose to work on the Fuzzer task (4b) and created a new branch (branch 2) to do this work on.
 - c. Jack chose to work on the Forensics task (4c), and once Dylan and Devin had finished their portions and uploaded to GitHub, Jack merged Branches 1 and 2 into the main branch, where he completed his task.
 - d. Further details about what each team member did within their section can be found below.
3. After completing the coding tasks in Activity 4, the team tested the full implementation of their code, capturing the output and writing a report to document their efforts and lessons learned.
 - a. During testing, changes were made to the fuzz.py file – the output format was changed to provide additional information, and the ability to run on GitHub Actions was added. Some methods chosen for fuzzing were presenting issues, so these were changed for other methods.

Git Hooks

The pre-commit hook runs off of a custom file-path so that it can be stored in the repo. To change to the new hook please run the following commands into your terminal.

```
git config advice.ignoredHook
```

```
git config core.hooksPath Hooks
```

The pre-commit file runs bandit against all python files in the codebase. The arguments I used were “q” so that the terminal is not flooded with info that is going to be stored into spreadsheet and instead just echos out that a report is being generated and when it successfully completes the report. Additionally, it echos out the file path of the report. The second argument I used was “r” so that it recursively goes through the entire repo selectively using a bash script to find the root node of the repository. I did this using “\$(git rev-parse --show-toplevel)” which allows bandit to run exclusively against the github repository. The argument “f” allowed me to format the report out to a CSV and I decided to include all information that bandit produces since all of it could be important. The output file is always generated in the same spot using the previously method of finding the root node of the repo. That way the hook always will work regardless of what operating system or machine it is run on.

Output

This shows the pre-commit file generating the report using bandit.

```
(kali㉿kali)-[~/Documents/test/BackLeft-SQA2022-AUBURN]
$ git commit -m test
Returning directory of repo
/home/kali/Documents/test/BackLeft-SQA2022-AUBURN
Creating report on all python files
Check outputs for generated spreadsheet
[main 657d3fd] test
1 file changed, 0 insertions(+), 0 deletions(-)
mode change 100644 => 100755 Hooks/pre-commit
```

This image shows a snippet of the log generated from bandit. Please check `.../outputs/bandit_outputs.csv` for the full spreadsheet.

```
bandit_output
outputs | bandit_output
1 filename,test_name,test_id,issue_severity,issue_confidence,issue_text,line_number,line_range,more_info
2 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,50,[50],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
3 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,83,[83],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
4 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,84,[84],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
5 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,124,[124],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
6 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,125,[125],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
7 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,126,[126],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
8 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,127,[127],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
9 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,160,[160],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
10 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,194,[194],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
11 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,195,[195],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
12 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,200,[200],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
13 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,215,[215],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
14 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/fuzz.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,215,[215],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
15 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/generation/probability_based_label_perturbation.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,28,[28],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
16 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/label_perturbation_attack/probability_based_label_perturbation.py,blacklist,B311,LOW,HIGH,Standard pseudo-random generators are not suitable for security/cryptographic purposes.,28,[28],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311
17 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/select_repos/dev_count.py,blacklist,B404,LOW,HIGH,Consider possible security implications associated with subprocess module.,7,[7],https://bandit.readthedocs.io/en/latest/blacklists/blacklist_imports.html#b404
18 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/select_repos/dev_count.py,start_process_with_partial_path,B607,LOW,HIGH,Starting a process with a partial executable path,26,[26],https://bandit.readthedocs.io/en/latest/plugins/b607_start_process_with_partial_path.html#b607
19 /home/kali/Documents/test/BackLeft-SQA2022-AUBURN/TestOrchestrator4M/TestOrchestrator4M-main/select_repos/dev_count.py,subprocess_without_shell_equals_true,B608,LOW,HIGH,subprocess call - check for execution of untrusted input.,26,[26],https://bandit.readthedocs.io/en/latest/plugins/b608_subprocess_without_shell_equals_true.html#b608
```

Lessons Learned

The main lesson I learned is that Github is an ever-changing ecosystem that requires a lot of work to understand and stay on top of. For instance, my main struggle was finding a way for the hook to be compatible between Linux and Windows. I had to research the new ways to find the root node of a repo as previously it was just an environment variable. This showed me that Github has changed a lot since I last had to use hooks. Additionally, I learned that running scripts is OS dependent which led me to finding a solution that works cross platform. The last lesson I learned is the importance of automated code testing to find vulnerabilities. It is much simpler to find obvious issues automatically instead of scouring the code to find them.

Fuzzer

Our fuzz.py file fuzzes 5 chosen Python methods from elsewhere in the project. The methods I chose to fuzz were primarily chosen because most of them did mathematical operations on the passed input without doing any type of check to ensure the correct type of input was given. Since most mathematical operations are not capable of being performed on many types of inputs, it was expected that passing these methods things such as strings, arrays, and unexpected number values would cause bugs to appear. The fuzzer runs through a few different random inputs on each chosen method to fuzz – the first is a randomly generated number, the second is a randomly generated string, and the third is a randomly chosen item from a list of known-useful fuzz inputs which was provided to us earlier in the class. To allow the fuzzer to run multiple times without stopping any time a crash is detected, each fuzz attempt

is within a try-except statement, which means that if the fuzzer causes the method to crash due to the input, then instead of stopping the program, it will instead print the values which caused the program to crash and continue.

Output

These pictures are the full Fuzz.py output

```
=====
Fuzzing Method: checkTestFile
=====
Failed! Method: checkTestFile. Input: -491104164
Exception:
expected str, bytes or os.PathLike object, not int
Total row: 0
Test: TEST 0.0
dtype: float64

Failed! Method: checkTestFile. Input: w6([aqQQ?7Jy\B'dS0? $"0P es|Y
Exception:
Cannot save file into a non-existent directory: '../Output'
Total row: 0
Test: TEST 0.0
dtype: float64

Failed! Method: checkTestFile. Input: AUX
Exception:
Cannot save file into a non-existent directory: '../Output'

=====
Fuzzing Method: cliffsDelta
=====
Failed! Method: cliffsDelta. Inputs: 37020965, -322984118
Exception:
object of type 'int' has no len()

Passed! Ran without error.

Passed! Ran without error.
```

```
=====
Fuzzing Method: eucDist
=====
Passed! Ran without error.

Failed! Method: eucDist. Inputs: h
EV'[0, :(X)|`.U2(gvzIITnTvdK9Fh~30}kZB
Exception:
unsupported operand type(s) for -: 'str' and 'str'

Failed! Method: eucDist. Inputs: "`"><script>\xE2\x80\x87javascript:alert(1)</script>, 0
Exception:
unsupported operand type(s) for -: 'str' and 'str'
```

```

1  ▶ Run python TestOrchestrator4ML/TestOrchestrator4ML-main/fuzz.py
2
3  =====
4  11 Fuzzing Method: checkTestFile
5  12 =====
6  13
7  14 Failed! Method: checkTestFile. Input: -23426647
8  15 Exception:
9  16 expected str, bytes or os.PathLike object, not int
10 17 Total row: 0
11 18 Test: TEST 0.0
12 19 dtype: float64
13
14 20
15 21 Failed! Method: checkTestFile. Input: V6jBasmK;8nglHMO/~/CFXTP
16 22 Exception:
17 23 Cannot save file into a non-existent directory: '../Output'
18 24 Total row: 0
19 25 Test: TEST 0.0
20 26 dtype: float64
21
22 27
23 28 Failed! Method: checkTestFile. Input: <script>alert(123)</script>
24 29 Exception:

```

Lessons Learned

The main lesson I learned from working on this section is how easy it is to crash a program by offering any form of unexpected input. True, most of the methods we are working with within this project do not expect the user to be directly inputting any information, but due to how Python typing works, it is still exceedingly easy to crash most of these methods by changing the expected input type to almost any other type. While this is not really surprising, it did make me think about how most programmers probably don't expect much variation when they are designing their code. The variation could be malicious, such as an attacker, or simply benign, as is the case with misinformed users, but if the variation is not accounted for, then it can lead to bugs, which could present vulnerabilities. I also had to learn a lot more about how GitHub actions work to successfully implement the python code running.

BUGS FOUND:

Almost all methods tested crash when given a non-expected input type. "runs" is the only method that did not crash with the given types inserted.

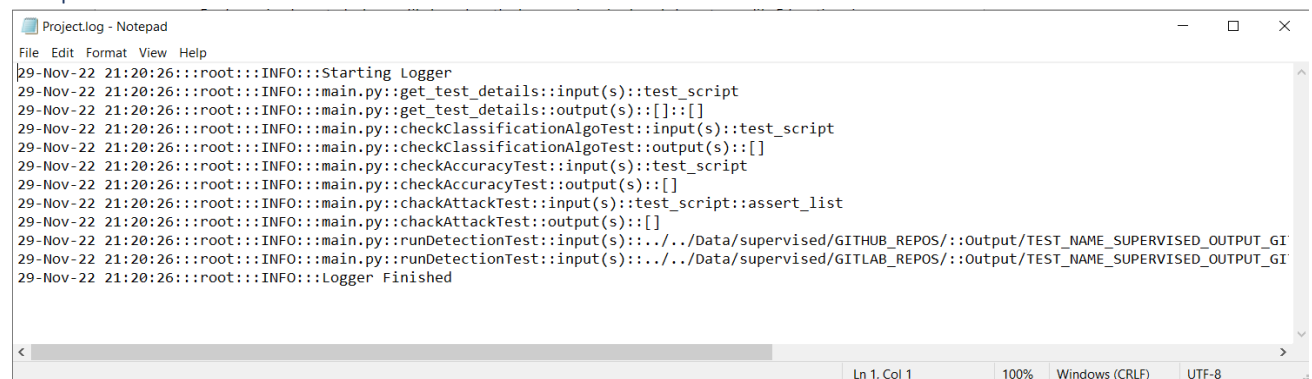
Other methods, such as CheckTestFile, do not seem properly configured to be run on any provided machine, as it uses relative file locations. Without changing the file path the program could not be run successfully.

Forensics

Jack Woods

For forensics, I created a logger file based on the loggers given in class. I chose to modify 5 functions in the main.py file under detection: get_test_details, checkClassificationAlgoTest, checkAccuracyTest, chackAttackTest, and runDetectionTest. I also modified __main__, but didn't know if that counted towards the 5 functions. I put a log statement at the beginning of each function to log the inputs and another statement at the end to log the outputs/results/return values. For the ease of my own understanding, I also created statements showing where the log begins and ends from each run, since the log file adds to itself instead of overriding what's there. The forensic output goes to a log file called "Project.log" in the TestOrchestrator4ML directory.

Output



```
Project.log - Notepad
File Edit Format View Help
29-Nov-22 21:20:26::root::INFO::Starting Logger
29-Nov-22 21:20:26::root::INFO::main.py::get_test_details::input(s)::test_script
29-Nov-22 21:20:26::root::INFO::main.py::get_test_details::output(s)::[]::[]
29-Nov-22 21:20:26::root::INFO::main.py::checkClassificationAlgoTest::input(s)::test_script
29-Nov-22 21:20:26::root::INFO::main.py::checkClassificationAlgoTest::output(s)::[]
29-Nov-22 21:20:26::root::INFO::main.py::checkAccuracyTest::input(s)::test_script
29-Nov-22 21:20:26::root::INFO::main.py::checkAccuracyTest::output(s)::[]
29-Nov-22 21:20:26::root::INFO::main.py::chackAttackTest::input(s)::test_script::assert_list
29-Nov-22 21:20:26::root::INFO::main.py::chackAttackTest::output(s)::[]
29-Nov-22 21:20:26::root::INFO::main.py::runDetectionTest::input(s)::../Data/supervised/GITHUB_REPOS::Output/TEST_NAME_SUPERVISED_OUTPUT_GI
29-Nov-22 21:20:26::root::INFO::main.py::runDetectionTest::input(s)::../Data/supervised/GITLAB_REPOS::Output/TEST_NAME_SUPERVISED_OUTPUT_GI
29-Nov-22 21:20:26::root::INFO::Logger Finished
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

From "Project.log"

Lessons Learned

Linux and Windows have a few compatibility issues. The original starter code for main.py output csv files using the directory “../Output...” which prevented my Windows laptop from finding the output directory. I fixed this by deleting the “../”, which may cause issues if the program is then tested on Linux. This problem did not occur with the “../Data/supervised...” lines. Also, Dylan taught me you can use ctrl + shift + p to fix a section of improperly indented code. I also learned several different syntax methods for formatting a logging statement, though those were also covered in class and simple.py. I pretty much learned how to use GitHub with GitHub Desktop through this project. I somehow managed to make it this far in software without ever creating or editing a GitHub repo, so I learned how to do all that here.